Campus Placement Prediction

▶ **Submitted to:**
Abdus Salam
Assistant Professor, CS

▶ **Submitted by:**
Group 16

20-43661-2	Kazi Rubab Bashar
20-43677-2	Shajid Kamal Joy

Dataset Description:

The dataset includes student placement records, offering insights into various factors that potentially influence employability outcomes. Each entry in the dataset is uniquely identified by a serial number, with accompanying details of personal information, educational achievements and employment-related parameters.

The gender column signifies the gender of each student, categorized as either Male (M) or Female (F). Educational performance is delineated through percentage scores in Secondary Education (ssc_p) and Higher Secondary Education (hsc_p), with corresponding information about the respective educational boards (ssc_b and hsc_b). Furthermore, the dataset captures the specialization pursued during Higher Secondary Education (hsc_s) and the type of undergraduate degree (degree_t), providing a clear view of academic backgrounds. Key indicators such as work experience (workex) and an employment test score (etest_p) shed light on additional dimensions of a student's profile. The dataset also provides details about MBA specialization (2pecialization), MBA percentage (mba_p) and most crucially, the placement status (status) of each student, denoted as 'Placed' or 'Not Placed'.

Task – 1:

It involves the initial step of loading the dataset into the analysis environment. The code utilizes the Pandas library to read the contents of the "data.csv" file into a DataFrame, labeled as 'df'.

- **Task 1:** Read/Load the dataset file in your program. Use Pandas library to complete this task.

```
# write task-1 solution

# start writing your code here

df = pd.read_csv("data.csv")
print(df)
```

Task – 2:

It handles data cleaning by dropping the 'salary' and 'sl_no' columns, streamlining the dataset for relevant features. The info() method is then employed to display essential information about the DataFrame, including data types and missing values. Then unique values and counts for

categorical columns are found for mapping dictionary. Then applied it to convert categorical values to numeric representations, enhancing data readiness.

- Df.drop() : Used for dropping the columns
- nunique() : Used to find number of unique value in columns
- unique() : Used to find the unique values
- df.replace() : Used Replace values in each specified column using the mapping dictionary

```
# write task-2 solution

# start writing your code here

column_to_drop = 'salary'
if column_to_drop in df.columns:
    df = df.drop(column_to_drop, axis=1)

column_to_drop = 'sl_no'
if column_to_drop in df.columns:
    df = df.drop(column_to_drop, axis=1)

df.info()

unique_values_count_column1 = df['gender'].nunique()
unique_values_count_column2 = df['ssc_b'].nunique()
unique_values_count_column3 = df['hsc_b'].nunique()
unique_values_count_column4 = df['hsc_s'].nunique()
unique_values_count_column5 = df['degree_t'].nunique()
unique_values_count_column6 = df['workex'].nunique()
unique_values_count_column7 = df['specialisation'].nunique()
unique_values_count_column8 = df['status'].nunique()


unique_values_column1 = df['gender'].unique()
unique_values_column2 = df['ssc_b'].unique()
unique_values_column3 = df['hsc_b'].unique()
unique_values_column4 = df['hsc_s'].unique()
unique_values_column5 = df['degree_t'].unique()
unique_values_column6 = df['workex'].unique()
unique_values_column7 = df['specialisation'].unique()
unique_values_column8 = df['status'].unique()
```

```
)

df.replace(mapping_dict, inplace=True)

print("DataFrame after replacing values:")
print(df)
```

Task – 3:

We used Matplotlib to create subplots for visualizing the frequency distributions of features in the dataset. The subplots function is used to set up a 2x7 grid of subplots, and the resulting array is flattened for simplicity. The code iterates through each column in the DataFrame to plotting histograms of the frequency distributions. Titles, labels and axis adjustments are applied for clarity.

- plt.subplots() : Used to set up the figure and axis for subplot
- axes.flatten() : Used to flatten the array of subplot to simplify indexing
- hist() : Used to plot histogram
- plt.tight_layout() : Used to adjust layout for better spacing
- plt.show() : Used to display the plots

```
# write task-3 solution

# start writing your code here

fig, axes = plt.subplots(nrows=2, ncols=7, figsize=(15, 8))


axes = axes.flatten()


for i, column in enumerate(df.columns):
    axes[i].hist(df[column], bins=15, color='black', edgecolor='blue')
    axes[i].set_title(f'F.D. of {column}')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Frequency')

for i in range(len(df.columns), len(axes)):
    axes[i].axis('off')


plt.tight_layout()


plt.show()
```

Task – 4:

The provided code produces two insightful visualizations to uncover relationships within the dataset. The first set of scatter plots, organized in a 2x7 grid, illustrates the interactions between the target variable ('status') and other numeric columns. Each subplot displays a scatter plot, providing a visual examination of potential correlations. The second visualization is a correlation heatmap, showcasing the correlation matrix for all numeric columns. This heatmap, employing a

'coolwarm' color scale, allows us to discern the strength and direction of relationships between various features.

- scatter() : Used for scatter plot
- corr() : Calculate the correlation matrix
- ax.pcolor() : used to a heatmap using Matplotlib's pcolor function
- plt.colorbar() : used to add colorbar

```
target_column = 'status'

for i, column in enumerate(df.columns):
    if df[column].dtype == 'float64' or df[column].dtype == 'int64':
        axes[i].scatter(df[column], df[target_column], color='black', edgecolor='skyblue', alpha=0.5)
        axes[i].set_title(f'{column} vs {target_column}')
        axes[i].set_xlabel(column)
        axes[i].set_ylabel(target_column)

for i in range(len(df.columns), len(axes)):
    axes[i].axis('off')


plt.tight_layout()


plt.show()
```

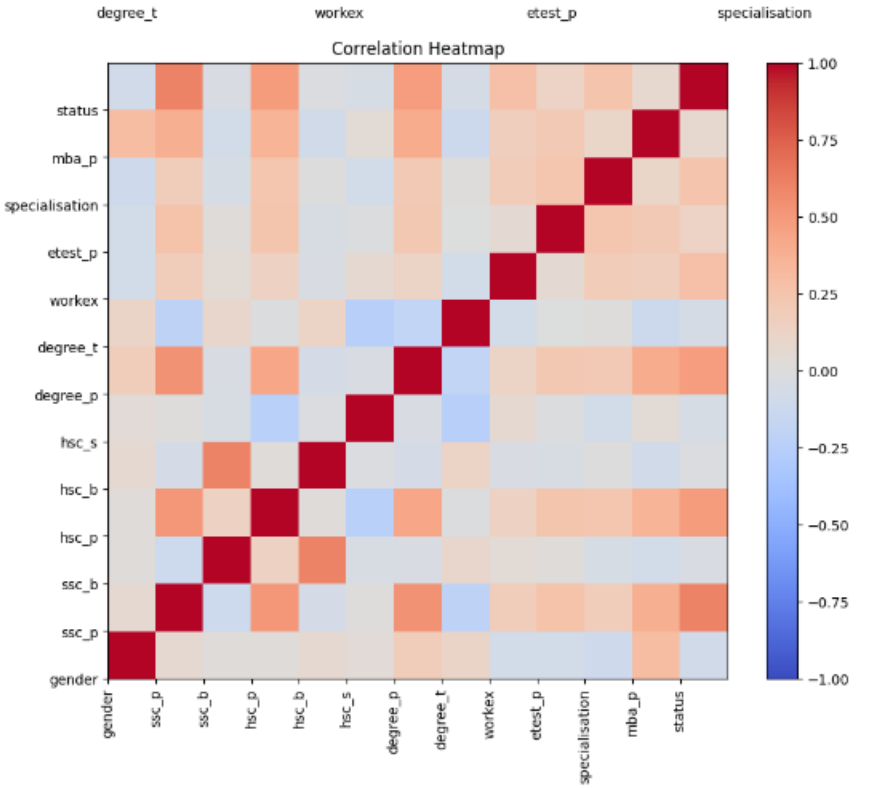
```
correlation_matrix = df.corr()
fig, ax = plt.subplots(figsize=(10, 8))

heatmap = ax.pcolor(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1)
cbar = plt.colorbar(heatmap)

ax.set_xticks(range(len(correlation_matrix.columns)))
ax.set_yticks(range(len(correlation_matrix.columns)))
ax.set_xticklabels(correlation_matrix.columns, rotation='vertical')
ax.set_yticklabels(correlation_matrix.columns)
plt.title('Correlation Heatmap')

plt.show()
```

Heatmap



Task – 5:

Here we did feature scaling to standardize the numeric columns in the dataset. Initially, only the columns with numeric data types (excluding the target column 'status') are selected. Subsequently, a StandardScaler instance is created, and it is applied to transform the selected numeric columns. The resulting DataFrame is then displayed, showcasing the dataset after the scaling process.

- StandardScaler(): Used to Create a StandardScaler instance
- fit_transform(): Used to apply scaling to the numeric columns

```
# write task-5 solution

# start writing your code here


numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
numeric_columns = numeric_columns[numeric_columns != 'status']

scaler = StandardScaler()

df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

print("DataFrame after scaling:")
print(df)
```

Task – 6:

Here the dataset was then divided into features (X) and the target variable (y). Using scikit-learn's train_test_split function, the data was split into training (80%) and testing (20%) sets.

- train_test_split():Used to split the data into training and testing sets
- sample():generates a random sample for the DataFrame
- reset_index():dropping the existing index and replacing it with a new one

```
df_numeric = df.select_dtypes(include=['float64', 'int64']).dropna()
df_numeric = df_numeric.sample(frac=1,random_state=5).reset_index(drop=True)


X = df_numeric.drop('status', axis=1)
y = df_numeric['status']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

Task – 7:

Here Naïve Bayes classifier (GaussianNB) was implemented using scikit-learn. The model was trained on the training set (X_train, y_train) and then used to predict the target variable on the testing set (X_test). Model performance was assessed using accuracy and the classification report, which provides metrics such as precision, recall and F1-score for each class.

- GaussianNB(): Used to create a Naïve Bayes classifier (GaussianNB)
- predict() : Used to make predictions on the testing set
- accuracy_score() : Used to Evaluate the performance of the classifier
- classification_report() : Used to make classification report

```
# write task-7 solution

# start writing your code here

nb_classifier = GaussianNB()

nb_classifier.fit(X_train, y_train)

y_pred = nb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_rep)
```

```
Accuracy: 0.7209302325581395

Classification Report:
                precision    recall  f1-score   support

           0       0.64      0.56      0.60        16
           1       0.76      0.81      0.79        27

    accuracy                           0.72        43
   macro avg       0.70      0.69      0.69        43
weighted avg       0.72      0.72      0.72        43
```

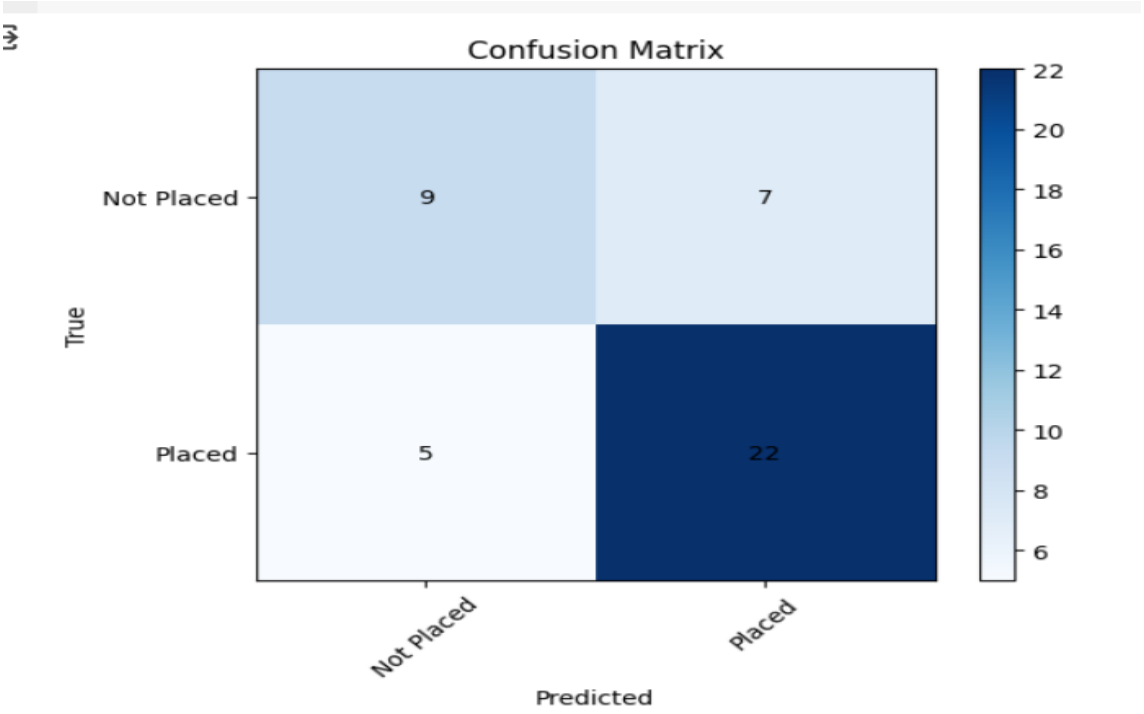
Task – 8:

The confusion matrix for the Naïve Bayes classifier's predictions on the testing set was calculated and visually represented using Matplotlib. The plot provides an insightful visualization of the model's performance by illustrating the counts of true positive, true negative, false positive, and false negative predictions.

- confusion_matrix() : Used to calculate the confusin matrix
- imshow(): Used to plot the confusion matrix

```
conf_matrix = confusion_matrix(y_test, y_pred)

plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
```



Task – 9:

Here various performance metrics were calculated to assess the Naïve Bayes classifier's effectiveness on the testing set. The accuracy, precision, recall and F1-score were computed using appropriate functions from the scikit-learn library. These metrics provide a comprehensive understanding of the model's ability to make correct predictions, especially in the context of binary classification.

- accuracy_score() : Calculate accuracy
- precision_score() : Calculate precision
- recall_score() : Calculate Recall
- f1_score() : Calculate f1 score

```python
accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.7209302325581395
Precision: 0.7586206896551724
Recall: 0.8148148148148148
F1 Score: 0.7857142857142857
```

Task – 10:

Here Naïve Bayes classifier (GaussianNB) was subjected to 10-fold cross-validation using the cross_val_score function from scikit-learn. The model's performance was evaluated over 10 folds and the accuracy for each fold was displayed. Additionally, the average accuracy across all folds was calculated too.

- GaussianNB():Used to create a Naïve Bayes classifier (GaussianNB)
- cross_val_score(): Used to perform 10-fold cross-validation

```python
nb_classifier = GaussianNB()

cv_scores = cross_val_score(nb_classifier, X, y, cv=10)

for i, score in enumerate(cv_scores, 1):
    print(f"Fold {i}: Accuracy = {score}")

average_accuracy = cv_scores.mean()
print("\nAverage Accuracy:", average_accuracy)
```

```
Fold 1: Accuracy = 0.8636363636363636
Fold 2: Accuracy = 0.8636363636363636
Fold 3: Accuracy = 0.8181818181818182
Fold 4: Accuracy = 0.8181818181818182
Fold 5: Accuracy = 0.7727272727272727
Fold 6: Accuracy = 0.9047619047619048
Fold 7: Accuracy = 0.8571428571428571
Fold 8: Accuracy = 0.8571428571428571
Fold 9: Accuracy = 0.7619047619047619
Fold 10: Accuracy = 0.6666666666666666

Average Accuracy: 0.8183982683982685
```